ARC integration in aCT and Python library for ARC CE REST

ARC7 technical workshop in Umeå
25-26 May 2023
Jakob Merljak

ARC REST integration in aCT

- ATLAS aCT setups switched from ARC Python bindings to ARC REST in April 2022
- speedup of operations across the board especially due to bulk requests
- support for multiple API versions; API 1.1 support in upcoming ARC7 is already used on the aCT test site using clusters

  arctestcluster-slurm-el{7,8,9}-arc7-ce1.cern-test.uiocloud.no

pyarcrest python library

- standalone python library for interaction with ARC REST extracted from and used by aCT and aCT client
- depends on ARC python bindings only for job description processing
- should be broadly applicable for integrations of ARC CE over REST using python

pyarcrest file structure

```
$ tree
.
├── README.md
├── setup.cfg
├── setup.py
└── src
    └── pyarcrest
        ├── arc.py
        ├── cli
        │   ├── arcrest.py
        │   └── __init__.py
        ├── common.py
        ├── errors.py
        ├── http.py
        ├── __init__.py
        └── x509.py
```

- *errors* module
- *http* module
- *x509* module
- *arc* module
- *arcrest* cli program

pyarcrest design

- *ARCRest* base class defining API interface and common functionality among API versions
  - one method for each operation exposed by ARC REST
  - higher-level methods for typical compound operations (e. g. job submission, creation and renewal of delegations)
  - “private” methods for internal reuse and organization
- Concrete class for every API version (e. g. *ARCRest_1_0*, *ARCRest_1_1*) overriding methods with version-specific behaviour

pyarcrest design

```
class ARCRest:
    def __init__(self, httpClient, apiBase="/arex", logger=getNullLogger()):
    def close(self):

    ### Direct operations on ARC CE ###
    def getAPIVersions(self):
    def getCEInfo(self):
    def getJobsList(self):
    def createJobs(self, description, delegationID=None, queue=None, isADL=True):
    def getJobsInfo(self, jobs):
    def getJobsStatus(self, jobs):
    def killJobs(self, jobs):
    def cleanJobs(self, jobs):
    def restartJobs(self, jobs):
    def getJobsDelegations(self, jobs):
    def downloadFile(self, url, path, blocksize=None):
    def uploadFile(self, url, path):
    def downloadListing(self, url):
    def getDelegationsList(self):
    def requestNewDelegation(self):
    def uploadDelegation(self, delegationID, signedCert):
    def getDelegationCert(self, delegationID):
    def requestDelegationRenewal(self, delegationID):
    def deleteDelegation(self, delegationID):

    ### Higher level job operations ###
    def createDelegation(self, lifetime=None):
    def renewDelegation(self, delegationID, lifetime=None):
    def submitJobs(self, delegationID, descs, queue, processDescs=True, matchDescs=True, uploadData=True, workers=10, blocksize=None, timeout=None):

    ### public static methods ###
    @classmethod
    def getAPIVersionsStatic(cls, httpClient, apiBase="/arex"):
    @classmethod
    def getClient(cls, url=None, host=None, port=None, proxypath=None, logger=getNullLogger(), blocksize=None, timeout=None, version=None, apiBase="/arex"):
```

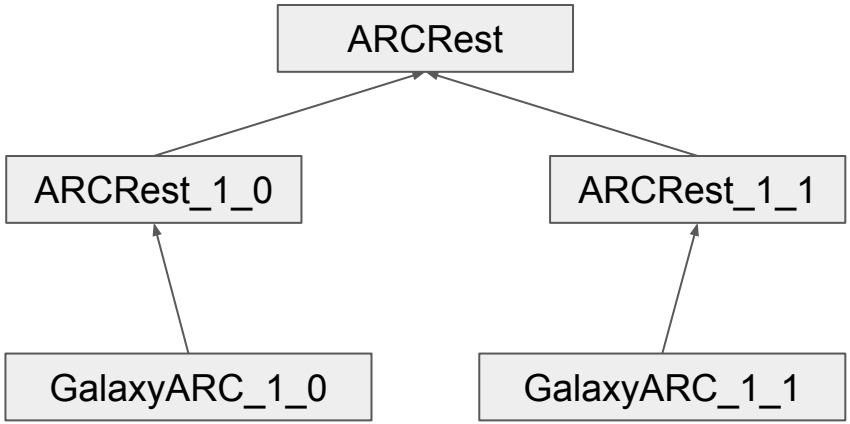
pyarcrest design

```
class ARCRest_1_0(ARCRest):

    def __init__(self, httpClient, apiBase="/arex", logger=getNullLogger()):
        # implementation specific to 1.0
        ...

    def createJobs(self, description, delegationID=None, queue=None, isADL=True):
        ...

    def submitJobs(self, delegationID, descs, queue, processDescs=True, matchDescs=True, uploadData=True, workers=10, blocksize=None, timeout=None):
        ...


class ARCRest_1_1(ARCRest):

    def __init__(self, httpClient, apiBase="/arex", logger=getNullLogger()):
        # implementation specific to 1.1
        ...

    def createJobs(self, description, delegationID=None, queue=None, isADL=True):
        ...

    def submitJobs(self, delegationID, descs, queue, processDescs=True, matchDescs=True, uploadData=True, workers=10, blocksize=None, timeout=None):
        ...
```

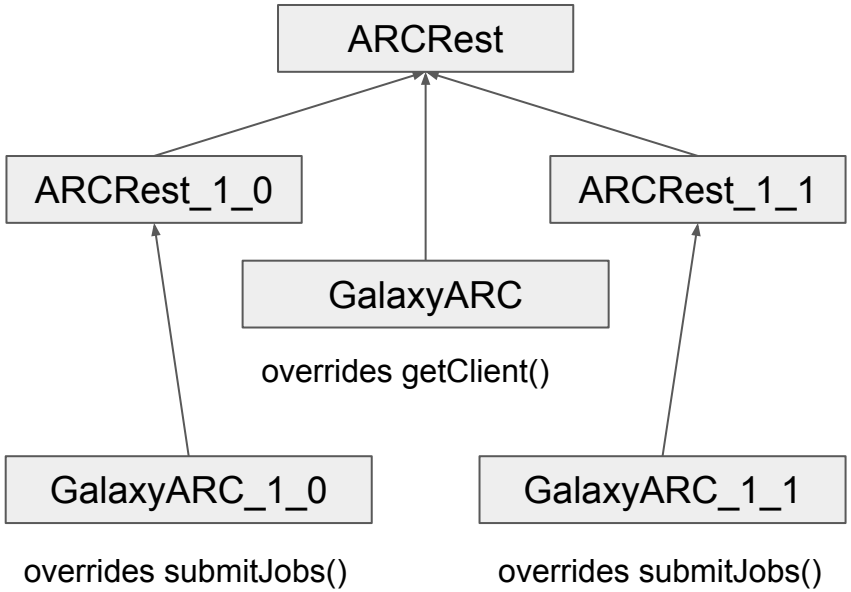
pyarcrest extension

- different potential ways to extend are available
  - if working with a single API version, the concrete API version class can be inherited from and instantiated for use
  - if working with multiple API versions, each concrete API version class can be inherited from
    - *getClient* method is used to pick the latest API version; it will be modified to reuse the functionality for inherited classes
    - one of the inherited classes needs to either reuse the API selection mechanism or come up with its own (based on requirements of integration)

pyarcrest extension example (Galaxy)



GalaxyARC_1_1.getClient(...)



GalaxyARC.getClient(...)

other inheritance graphs are possible

API showcase

```
>>> arcrest.getAPIVersions()
['1.0', '1.1']

>>> arcrest.getCEInfo()
{'Domains': {'AdminDomain': {'ID': 'urn:ad:UNDEFINEDVALUE', ...}, ...}, ...}

>>> arcrest.getJobsList()
['f26dc52530d3', '65386daeb98e', '8695212bee7b']

>>> arcrest.getDelegationsList()
[{'id': 'f9703e408e4c', 'type': 'x509'}, ...]

>>> csr, delegid = arcrest.requestNewDelegation()
>>> csr
'-----BEGIN CERTIFICATE REQUEST-----\nMIICRTCCAS0CAQIwADCCASIw...'
>>> delegid
'a06989ab1a34'
>>> arcrest.uploadDelegation(delegid, arcrest._signCSR(csr))
>>>

>>> csr = arcrest.requestDelegationRenewal(delegid)
>>> arcrest.uploadDelegation(delegid, arcrest._signCSR(csr))
>>>

>>> arcrest.deleteDelegation(delegid)
>>>
```

```
>>> arcrest.createDelegation()
'a06989ab1a34'
```

```
>>> arcrest.renewDelegation(delegid)
>>>
```

Two alternative approaches to API

```
jobids = ['123', 'abc', ...]
jobs = [ARCJob(id=id) for id in jobids]
arcrest.getJobsInfo(jobs)
for job in jobs:
    if not job.errors:
        exitcode = job.ExitCode
    else:
        for error in job.errors:
            if isinstance(error, ARCHTTPError):
                # process ARC error
            elif isinstance(error, SomeOtherException):
                # process another type of error
            elif isinstance(result, Exception):
                # process any exception
```

```
jobids = ['123', 'abc', ...]
results = arcrest.getJobsInfo(jobids)
for jobid, result in zip(jobids, results):
    if isinstance(result, ARCHTTPError):
        # process ARC error
    elif isinstance(result, SomeOtherException):
        # process another type of error
    elif isinstance(result, Exception):
        # process any exception
    else:
        exitcode = result.get("ExitCode", None)
```

Two alternative approaches to API

```
class ARCJob:

    def __init__(self, id=None, descstr=None):
        self.id = id
        self.descstr = descstr
        self.name = None
        self.delegid = None
        self.state = None
        self.errors = []
        self.downloadFiles = []
        self.inputFiles = {}

        self.ExecutionNode = None
        self.UsedTotalWallTime = None
        self.UsedTotalCPUTime = None
        self.RequestedTotalWallTime = None
        self.RequestedTotalCPUTime = None
        self.RequestedSlots = None
        self.ExitCode = None
        self.Type = None
        self.LocalIDFromManager = None
        self.WaitingPosition = None
        self.Owner = None
        self.LocalOwner = None
        self.StdIn = None
        self.StdOut = None
        self.StdErr = None
        self.LogDir = None
        self.Queue = None
        self.UsedMainMemory = None
        self.SubmissionTime = None
        self.EndTime = None
        self.WorkingAreaEraseTime = None
        self.ProxyExpirationTime = None
        self.RestartState = []
        self.Error = []
```

Two alternative approaches to API

```
def getJobsList() -> [ARCJob]
def createJobs([ARCJob]) -> None
def getJobsInfo([ARCJob]) -> None
def getJobsStatus([ARCJob]) -> None
def killJobs([ARCJob]) -> None
def cleanJobs([ARCJob]) -> None
def restartJobs([ARCJob]) -> None
def getJobsDelegations([ARCJob]) -> None
```

```
def getJobsList() -> [jobid]
def createJobs(description) -> [(jobid, state)]
def getJobsInfo(jobids) -> [infoDict | exception]
def getJobsStatus(jobids) -> [status | exception]
def killJobs(jobids) -> [True | exception]
def cleanJobs(jobids) -> [True | exception]
def restartJobs(jobids) -> [True | exception]
def getJobsDelegations(jobids) -> [[delegid] | exception]
```

Two alternative approaches to API

- use of ARCJob objects
- call operation on objects, pick up the values from them

- tradeoffs:
  - imposes a particular data structure

- use of individual values
- call operation on values and process the returned results

- tradeoffs:
  - requires handling proper alignment of results to parameters and parameters to other parameters

Personal note: the base API is probably cleaner without ARCJob objects. Structuring data becomes more beneficial for higher level methods, like *submitJobs*, *uploadJobFiles*, *downloadJobFiles*. Also, the library user might want to structure data more specifically to their context which can make dealing with imposed structure more awkward.

Matchmaking API

- REST client is supposed to do matchmaking
- the REST specification does not mention this
- the matchmaking that the REST client is supposed to do has to be systematically documented
  - what has to be matched (e. g. runtime environment)
  - how does it have to be matched (for e. g. runtime: string matching? Specific versioning scheme(s)?)
- personal note: legacy or obscure functionality should be omitted to improve usability
- currently implemented: walltime, runtime environment (string match), queue

Token auth

- currently, proxy certs assumed for delegations functionality and token based delegations is not implemented; this should not be difficult to add, though

- token support was worked/experimented with for Galaxy integration?
- curious: is it possible to use token delegations without authenticated HTTPS connection using proxy cert?

Job description processing

- required to properly support xRSL (user-side to server-side xRSL conversion has to be done by client)
- currently, this is done using ARC python bindings
- supporting *all* xRSL is difficult, so it will probably never be implemented in pure Python
- ADL is better but would still require duplicating the effort

- a standalone, portable, low dependency library available on all platforms implementing both xRSL and ADL would be nice :)
- even more preferably, a more convenient job description language (YAML schema?)

Proxy certificates

- ARC client required for generating proxy certificate from user cert
- we experimented with python implementation of proxy certificate generation but the difficult part is to implement VOMS attributes (no obvious libraries or documentation)

Links

https://github.com/jakobmerljak/pyarcrest

Will be moved at least to ARCControlTower Github organization.

pip install git+https://github.com/jakobmerljak/pyarcrest@dev

pip install git+https://github.com/jakobmerljak/pyarcrest@altapi